# OpenGL: Graphics Programmability

## Session 504

# OpenGL: Graphics Programmability

**Geoff Stahl and James McCombe**
**OpenGL Engineering**

# What You Will Learn

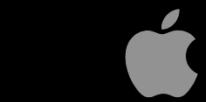
- “Shade Trees” and beyond
- What is graphics programmability?
- Vertex Programming
- OpenGL Shader Builder Tutorial
- Texture/Fragment Shading

# Shade Trees and Beyond

- "Shade Trees" [Cook] 1984
- RenderMan "Luxo Jr." [Pixar] 1986
- Chameleons and Werewolves [NVIDIA] 2001
- Stanford Shading Language, "Doom 3" and beyond

# Chameleon Up Close

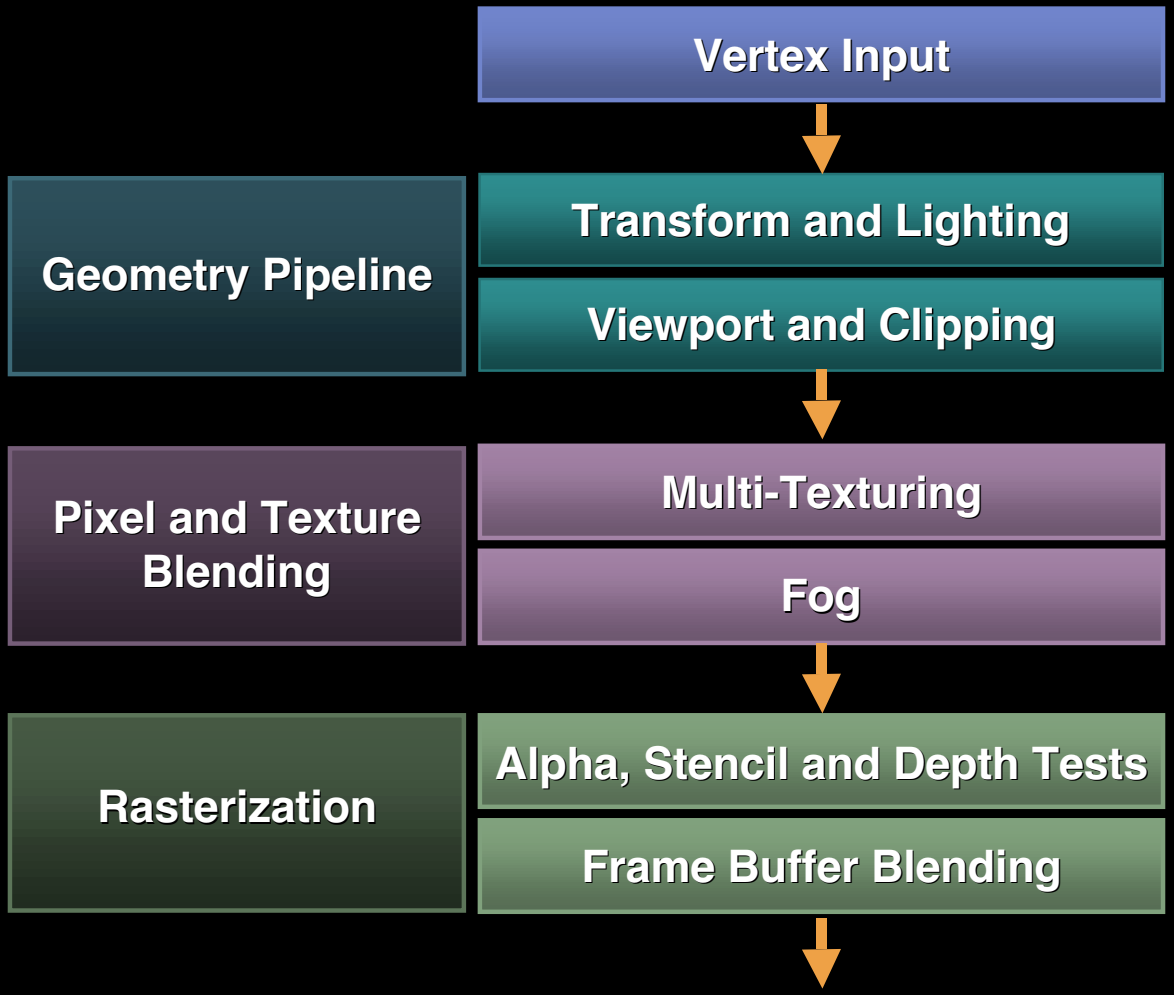
- Skinned vertex mesh
- Bump mapped
- Gloss map
- Cubic environment map
- Refraction
- Fogging



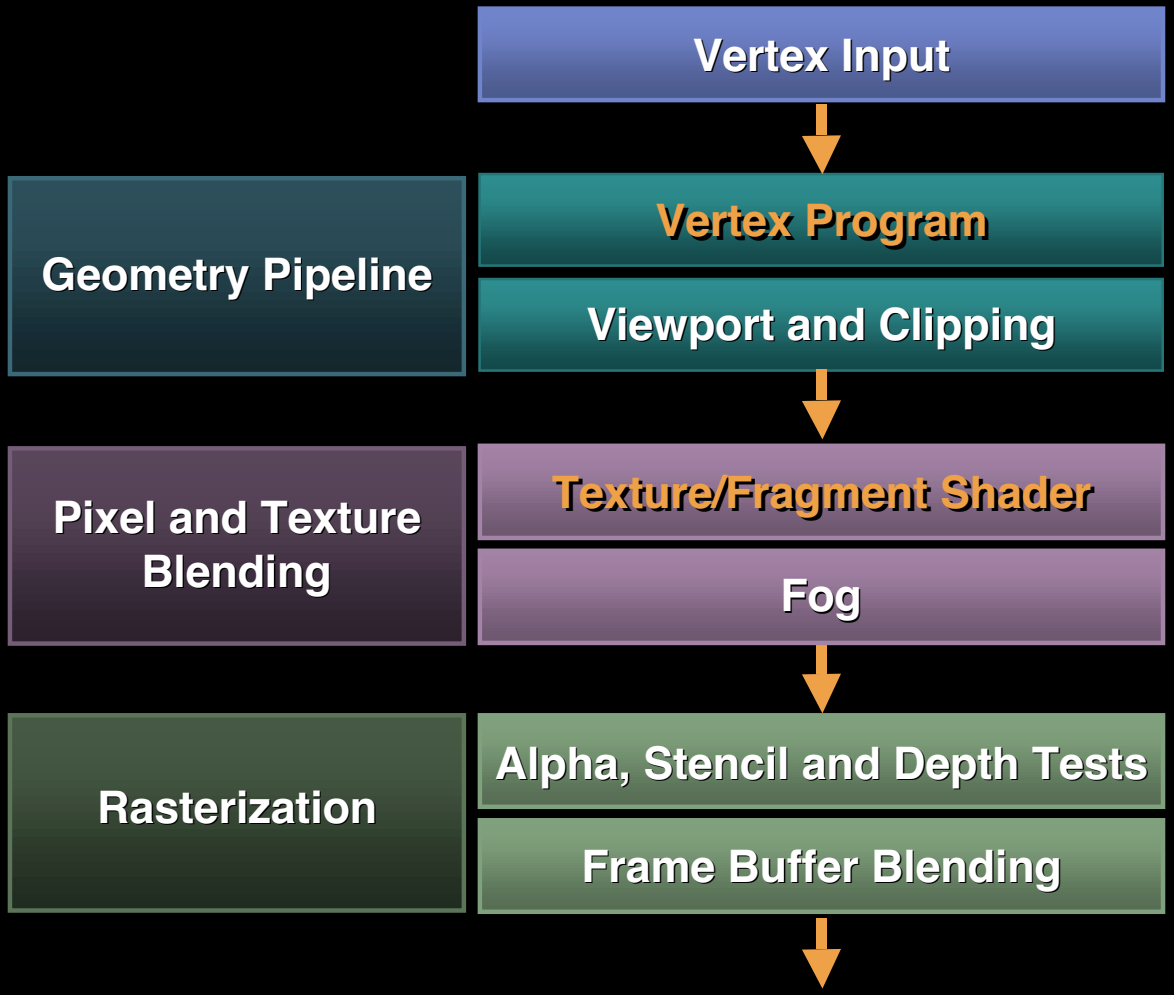
# What Is Programmability?

- Traditional graphics pipeline
- Programmable pipeline
- Where are we now?

# Traditional Graphics Pipeline




# Traditional Graphics Pipeline



# Where Are We Now?

- Mac OS X Jaguar targets
  - ARB Vertex Program
  - NVIDIA
    - Texture Shader, Texture Shader 2, Texture Shader 3, Register Combiners, Register Combiners 2
  - ATI
    - Fragment Shader
- Hardware Support
  - NVIDIA GeForce 3 and 4, ATI Radeon 8500
- Optimized Software Vertex Programs
  - All other hardware


# ARB Vertex Program

- Target for Mac OS X Jaguar
  - Currently with ARB working group
    - 84 issues (76 resolved, 8 unresolved)
- Seed has snapshot
  - Some syntactic differences
  - Functionality is almost exact
- OpenGL Shader Builder
  - Updated for final extension

# Vertex Programs

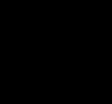
- Replaces fixed function transform and lighting operations
- Responsible for . . .
  - Transforming vertices to clip space
  - Per-vertex lighting
  - Texture coordination generation
  - Normal transformation, scaling, and normalization

# Why Vertex Programs?

- Transfers per vertex ops to GPU
    - Performance is on par with fixed pipe
- Handle non-standard pipeline functionality
    - Light models
    - Skinning
    - Non-standard vertex data
- Parametric control of vertex pipe
- Efficiency

# Computation Model

- Vector engine
  - Operations are 4 component float, 32 bits each
- Single vertex
  - No vertex generation or primitives
- State machine
  - Inputs modified only by invoking application
  - No vertex interdependence
- No branching
  - Evaluate and select instead
- Output data required by current rasterization model



# Input

- Vertex attributes
  - Conventional
    - Position, normal, color, fog coordinate, weight
    - Texture coordinates
    - Material properties
  - Generic attributes
  - Limited amount can be used per program
    - **MAX_VERTEX_ATTRIBS_ARB**
      - 16 minimum
  - Read only

# Input . . .

- Vertex program parameters
  - Defined outside Begin/End
  - OpenGL state
  - User data
    - Program local parameters
    - Program environment parameters
  - **MAX_VERTEX_PROGRAM_PARAMETERS_ARB**
    - 96 minimum
  - Read only

# ARB Example Input

```
ATTRIB iPos         = vertex.position;
ATTRIB iNormal      = vertex.normal;
ATTRIB specExp      = material.shininess;
ATTRIB ambMat       = material.ambient;
ATTRIB diffMat      = material.diffuse;
ATTRIB specMat      = material.specular;

PARAM  mvinv[4]     = { state.matrix.modelview.invtrans };
PARAM  mvp[4]       = { state.matrix.mvp };
PARAM  lightDir     = state.light[0].position;
PARAM  halfDir      = program.env[0];
PARAM  specExp      = program.env[1];
PARAM  vec001       = { 0, 0, 1 };
```

# Local Variables

- Vertex program temporaries
  - 4 component 32 bit floats
    - Initial value undefined
  - Declared explicitly
  - MAX_VERTEX_PROGRAM_TEMPORARIES_ARB
    - 12 Minimum
  - Read and write

```
TEMP   xfNormal, temp, dots, tmpColor;
```

# Output

- Vertex program results
  - Position in clip coordinates (required)
  - Color (front/back, primary/secondary)
  - Fog coordinate
  - Point size
  - Texture coordinates
- Must output all results required for current rasterization model
- Write only

# ARB Example Output

```
OUTPUT oPos             = result.position;
OUTPUT oColor           = result.color;
OUTPUT oBackSecColor    = result.color.back.secondary;
OUTPUT oFogCoord        = result.fog;
```

# Instruction Set Overview

- Assembly language
  - Graphically oriented
  - Source swizzle and negate
  - Write mask
  - **MAX_VERTEX_PROGRAM_INSTRUCTIONS_ARB**
    - Minimum 128 operations
  - Most instructions are single operation
    - Some maybe compound
      - Noted in specification and implementation dependent

# ARB Instruction Set

- ARL
- MOV, MUL, ADD, SUB, MAD, DST, SWZ
- DP3, DP4, XPD
- MIN, MAX, SLT, SGE, ABS
- FLR, FRC
- RCP, RSQ, POW, LOG, LG2, EXP, EX2, LIT

# ARB Example

```
!!ARBvp1.0
ATTRIB  iPos            = vertex.position;
ATTRIB  iNormal         = vertex.normal;
PARAM   mvinv[4]        = { state.matrix.modelview.invtrans };
PARAM   mvp[4]          = { state.matrix.mvp };
PARAM   lightDir        = program.env[0];
PARAM   halfDir         = program.env[1];
PARAM   specExp         = program.env[2];
PARAM   ambientCol      = program.env[3];
PARAM   diffuseCol      = program.env[4];
PARAM   specularCol     = program.env[5];
TEMP    xfNormal, temp, dots;
OUTPUT oPos             = result.position;
OUTPUT oColor           = result.color;
```

# Example . . .

```
DP4   oPos.x, mvp[0], iPos;
DP4   oPos.y, mvp[1], iPos;
DP4   oPos.z, mvp[2], iPos;
DP4   oPos.w, mvp[3], iPos;
DP3   xfNormal.x, mvinv[0], iNormal;
DP3   xfNormal.y, mvinv[1], iNormal;
DP3   xfNormal.z, mvinv[2], iNormal;
DP3   dots.x, xfNormal, lightDir;
DP3   dots.y, xfNormal, halfDir;
MOV   dots.w, specExp.x;
LIT   dots, dots;
MUL   temp, dots.x, ambientCol;
MAD   temp, dots.y, diffuseCol, temp;
MAD   oColor.xyz, dots.z, specularCol, temp;
END
```

# ARB OpenGL Interface

```
uint program;
GenProgramsARB (1, * program);
BindProgramARB (GL_VERTEX_PROGRAM_ARB, program);
ProgramStringARB (GL_VERTEX_PROGRAM_ARB,
   PROGRAM_FORMAT_ASCII_ARB, strlen (string), string);
...
glContextParametervARB (GL_VERTEX_PROGRAM_ARB,
   param, 4, GL_FLOAT, (float*)&paramVec);
...
   glVertexAttrib4fvARB(1, &attribVec);
   glVertex3f (1.0, 1.0, 1.0);
...
DeleteProgramsARB (1, program);
```

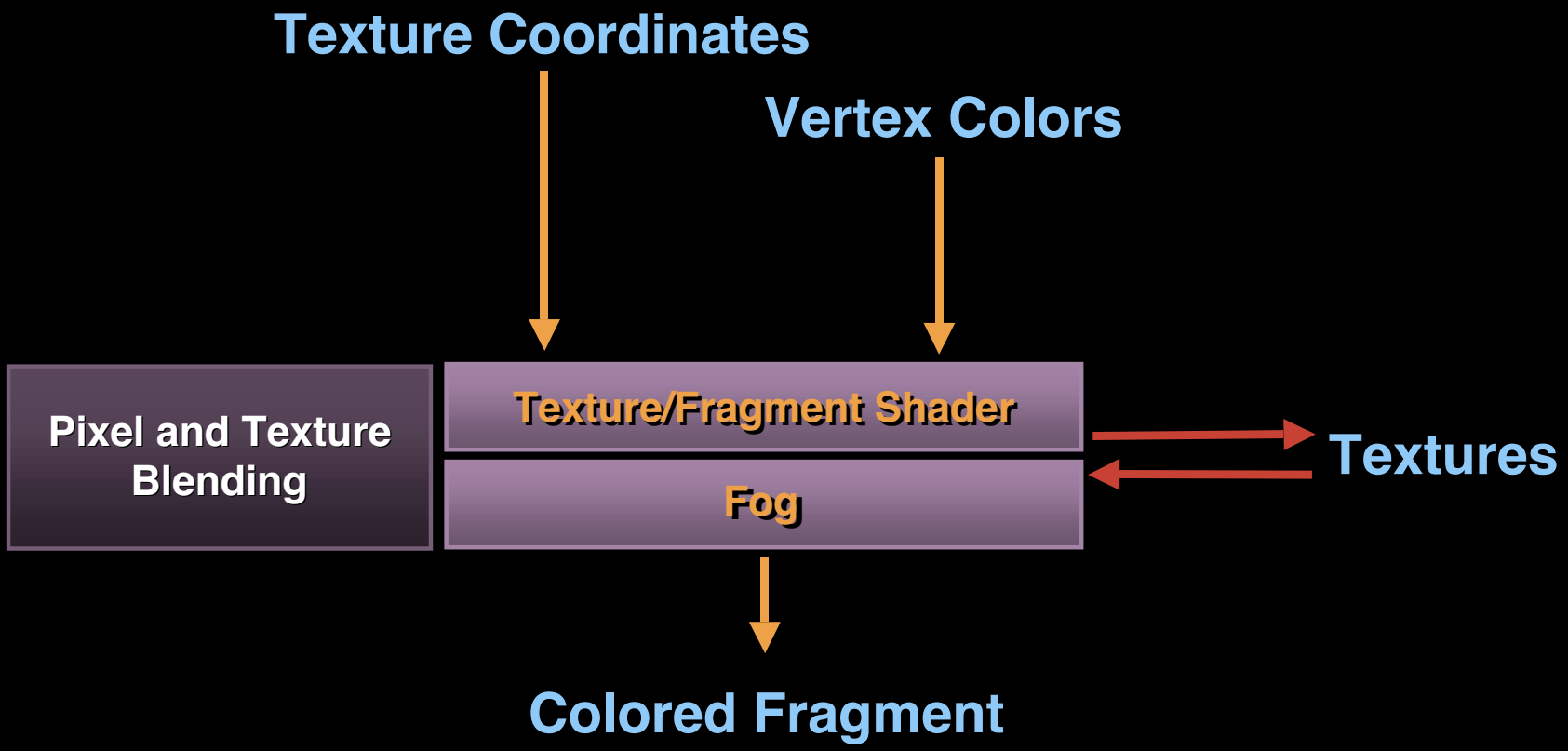
# Demo


**OpenGL Shader Builder**

# Texture/Fragment Shaders

- What are Texture/Fragment Shaders?
  - Per fragment texture address look up and multi-texture combine
  - Determine texture address
  - Perform texture look up(s)
  - Combine texel colors for fragment color
  - Output fragment color data

# Texture/Fragment Shaders . . .

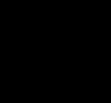


# Texture/Fragment Shaders . . .

- Why not a single Extension?
- Vendor Specific Extensions
    - Texture Shader and Register Combiners
    - Fragment Shader

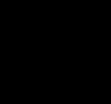

# Texture Shader

- Provides texture address look up via shader stages
- 3 extensions provide 37 canned shader operations
  - Conventional
    - 1D, 2D, 3D,texture rectangle, cubemap
  - Special case
    - None, pass through, cull fragment
  - Simple dependent texture fetches
    - Result of one texture lookup affects subsequent texcoords
  - Dot product dependent textures fetches



# Register Combiners

- Extremely configurable fragment coloring
- Replaces texture environment, color sum, and fog
    - Non-linear, all results available to all combiners
- 2 Extensions
    - Register Combiners 2 adds addition color constant flexibility
- NVIDIA Specific




# Fragment Shader

- Unified instruction set
- Controls both texture address and color data
  - So provides functionality of both Texture Shader and Register Combiners
- Similar in layout to ARB Vertex Program
- ATI Specific
  - Radeon 8500 and future

# Fragment Shader Example

- Bumped cubic environment mapped shader
  - Based on a very preliminary spec

```
// Routing instructions
RESULT result = OUTPUT COLOR;
LOCAL texCoord0 = TEX_COORD 0 SWIZZLE_STR;
LOCAL texCoord1 = TEX_COORD 1 SWIZZLE_STR;
LOCAL texCoord2 = TEX_COORD 2 SWIZZLE_STR;
LOCAL texCoord3 = TEX_COORD 3 SWIZZLE_STR;
LOCAL texCoord3 = TEX_MAP 4 TEX_COORD 5 SWIZZLE_STR;
```

# Fragment Shader Example . . .

```
// Begin first clause of program
// Three dot products go from tangent space to cube map space
COLOR_DOT3 texCoord0.r NONE texCoord0 NONE texCoord4 NONE;
COLOR_DOT3 texCoord0.g NONE texCoord1 NONE texCoord4 NONE;
COLOR_DOT3 texCoord0.b NONE texCoord2 NONE texCoord4 NONE;

// 2 * (N dot Eye)
COLOR_DOT3 texCoord2 MOD.2X texCoord0 NONE texCoord3 NONE;
// 2 * N * (N dot Eye)
COLOR_MUL  texCoord2 NONE texCoord0 NONE texCoord2 NONE;
// N dot N
COLOR_DOT3 texCoord1 NONE texCoord0 NONE texCoord0 NONE;
// 2 * N * (N dot Eye) - Eye * (N dot N)
COLOR_MAD  texCoord1 NONE texCoord3 MOD.NEG texCoord1 NONE
           texCoord2 NONE;
```

# Fragment Shader Example . . .

```
// Pass 2 -Three "dependent reads"
// Sample diffuse cubic env map
LOCAL texCoord0 = TEX_MAP 0 texCoord0 SWIZZLE_STR;
// Sample specular cubic env map
LOCAL texCoord1 = TEX_MAP 1 texCoord1 SWIZZLE_STR;
// Sample the base map (gloss in a)
LOCAL texCoord2 = TEX_MAP 2 TEX_COORD 5 SWIZZLE_STQ;

// Begin second clause of the program
// You can have up to 8 instructions here too
// diffuse * base
COLOR_MUL  texCoord0 NONE texCoord0 NONE texCoord2 NONE;
// (diffuse * base) + (spec * gloss)
COLOR_MAD  texCoord0 NONE texCoord0 NONE texCoord2.a NONE
           texCoord1 NONE;
COLOR_MOV  result NONE texCoord0 NONE;
```

# Demo

**Wolfman**
[NVIDIA 2002]

# Deconstructing Wolfman

- Skinned mesh
  - Built from static vertex array and weighted bone structure
- Rendered fur
  - Constructed from concentric, semi-transparent textured shells and fins
    - Based on research by Jed Lengyel, Microsoft
- Shadows volumes
  - **SGIX_depth_texture and SGIX_shadow**
- Bump mapping
  - Per pixel bump mapping

# References

- ARB Vertex Program Specification
  - ARB working group
- NVIDIA and ATI web sites
- On-line

  **<http://www.opengl.org>**
  **<http://lists.apple.com/mailman/listinfo/mac-opengl>**
  **<http://developer.apple.com/opengl>**

# Wrap Up

- Programmability
- Vertex Programs
- OpenGL Shader Builder
    - On seed CD
- Texture/Fragment Shaders

# Roadmap

| | |
|---|---|
| **500 Graphics and Imaging Overview** | Room A2<br>**Tue., 10:30am** |
| **503 Exploring the Quartz Compositor** | Hall 2<br>**Tue., 3:30pm** |
| **504 OpenGL:<br>Graphics Programmability** | Room A2<br>**Tue., 5:00pm** |
| **505 OpenGL: Integrated Graphics I** | Room J<br>**Wed., 9:00am** |

# Roadmap

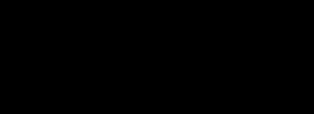
| | |
|---|---|
| **506 OpenGL: Integrated Graphics II** | Room J<br>**Wed., 10:30am** |
| **509 ColorSync and Digital Media** | Room C<br>**Wed., 5:00pm** |
| **511 Games Solutions: Graphics, Events, and Tidbits** | Room C<br>**Thurs., 10:30am** |
| **512 Games Solutions: NetSprocket and OpenPlay** | Room C<br>**Thurs., 2:00pm** |

# Roadmap

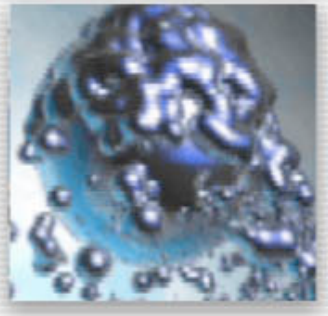
| | |
|---|---|
| **513 OpenGL: Advanced 3D** | Room J<br>**Thurs., 3:30pm** |
| **514 OpenGL:<br>Performance and Optimization** | Room J<br>**Thurs., 5:00pm** |
| **516 Graphics and Imaging<br>Performance Tuning** | Hall 2<br>**Fri., 3:30pm** |
| **FF018: Graphics and Imaging** | Room J1<br>**Fri., 5:00pm** |

# Who to Contact

**Sergio Mello**
3D Graphics Technology Manager
**sergio@apple.com**



# Q&A

**Sergio Mello**
**3D Graphics Technology Manager**
**Worldwide Developer Relations**
**sergio@apple.com**

**http://developer.apple.com/wwdc2002/urls.html**


WWDC2002


WWDC2002


WWDC2002